REMARKS/ARGUMENTS

Claims 1-38 are pending in the present application.

This Amendment is in response to the final Office Action mailed August 6, 2004. In the final Office Action, the Examiner requested correction to minor misspellings in Specification, and rejected claims 1-38 under 35 U.S.C. §103(a). Reconsideration in light of the and remarks made herein is respectfully requested.

Specification

1. In the final Office Action, the Examiner has noted that in multiple locations of the Specification, the word "field" is misspelled as "filed" and correction is requested. In response, Applicant has corrected the misspelling accordingly.

Rejection Under 35 U.S.C. § 103

1. In the Office Action, the Examiner rejected claims 1-38 under 35 U.S.C. §103(a) as being unpatentable over U.S. Pre Grant Publication 2001/0047510 issued to Angel et al. ("Angel") in view of "Poor Man's Watchpoints", by Max Copperman and Jeff Thomas (1995) ("Copperman"). Applicant respectfully traverses the rejection and contends that the Examiner has not met the burden of establishing a prima facie case of obviousness. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *MPEP §2143, p. 2100-129 (8th Ed., Rev. 2, May 2004).* Applicants respectfully contend that there is no suggestion or motivation to combine their teachings, and thus no *prima facie* case of obviousness has been established.

Applicants reiterate the arguments set forth in the previously filed Response to the Office Action.

Angel discloses a byte code instrumentation. A technique to instrument a byte code program includes examining the byte code, selecting portions of the byte code for

instrumentation, and instrumenting the portions to provide instrumented byte code (Angel, paragraph [0014]). Memory access instructions are instrumented to detect illegal memory operations at runtime (Angel, paragraph [0091]. In addition, exiting and entering blocks of code where variables become defined and undefined are monitored (Angel, paragraph [0091]).

Copperman discloses a technique to implement watch points using code patching. When the user sets a watch point, the debugger sets the register $fp to point to a register save area in the debuggee's static data space. When no watch points are set, the first instruction in the patch branches around the rest of the patch if $fp contains (Copperman, page 38, third paragraph under section "The Debuggee").

Angel and Copperman, taken alone or in any combination, does not disclose, suggest, or render obvious (1) compiling a function including a byte code sequence, (2) generating an instrumentation code corresponding to a field watch, (3) guarding execution of the instrumentation code if the field watch is not activated; and (4) inserting the instrumentation code to the native code. There is no motivation to combine Angel and Copperman because neither of them addresses the problem of compilation according to a field watch. There is no teaching or suggestion that guarding execution of instrumentation code or a field byte code accessing or modifying the field is present. Angel, read as a whole, does not suggest the desirability of generating an instrumentation code corresponding to the field watch.

The Examiner states that Angel discloses, generating an instrumentation code corresponding to a field watch of the field (Final Office Action, page 3). Applicant respectfully disagrees. The cited paragraph, [0125], merely states that "[d]escribed below are methods of automatically editing the executable byte code representation...for generating instrumented byte code" (Angel, page 11, paragraph [0125]). There is no teaching or suggestion on the use of a field watch in generating instrumentation code. A field watch sequence may include instruction sequence to spill the mimic stack operands, which are live at the field access point, to their canonical spill locations (See, for example, Specification, page 14, paragraph [0047]).

The Examiner further states that Copperman discloses guarding execution of the instrumentation code if the field watch is not activated by disclosing setting or not setting the watchpoints, or entering or enabling a watchpoint command (Final Office Action, page 4).

Applicant respectfully disagrees. A watchpoint or watchpoint command is not the same as a field watch as discussed above.

In the final Office Action, the Examiner responds to the Applicant's arguments (final Office Action, pages 10-14). Applicant contends that the Examiner's responses failed to overcome Applicant's arguments.

The Examiner states that Angel and Copperman are justifiably combined as both and related to instrumentation of code in order to facilitate debugging (Final Office Action, page 11, item (A)). However, neither Angel nor Copperman discloses or suggests a field byte code that accesses or modifies a field and generating an instrumentation code corresponding to a field watch of the field. Therefore, the combination of Angel nor Copperman is improper.

The Examiner further states that compilation according to a field watch is not a claim limitation (Final Office Action, page 12, item (C)). Applicant respectfully disagrees. Applicant contends that neither Angel nor Copperman addresses the problem of compilation according to a field watch in that there is an operation of generating an instrumentation code corresponding to a field watch of the field.

The Examiner further states that an instruction being instrumented could be a field watch, i.e., related to memory variable access (Final Office Action, page 13, item (D)). Applicant respectfully disagrees. A memory variable access is not a field watch. A field is a Java variable that is defined in a Java object. Field access events are generated when the field specified is about to be accessed. Field access from Java Language Code or from NJI are watched (See Specification, page 12, paragraph [0039].

The Examiner further states that a Web definition states a watchpoint is a type of breakpoint that is triggered whenever the class field being monitored is modified. However, the Examiner did not produce evidence of this definition. Furthermore, even if that definition is correct, it does not mean that Copperman discloses or suggests the watchpoint in that context. Copperman discloses that watchpoints aid in finding bugs that caused by an assignment though an unidentified pointer modifying an arbitrary memory location (Copperman, page 37, first paragraph under section heading "Introduction"). Therefore, a watchpoint in Copperman is not the same as the field watch.

The Examiner further offers the definition from "JAVA Debug Interface". However, there is no linkage between this definition and Copperman.

Therefore, Applicant believes that independent claims 1, 16, 31 and their respective dependent claims are distinguishable over the cited prior art references. Accordingly, Applicant respectfully requests the rejections under 35 U.S.C. §112 and 35 U.S.C. §103(a) be withdrawn.

Conclusion

Applicant respectfully requests that a timely Notice of Allowance be issued in this case.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP

Dated: September 29, 2004 By _____

Thinh V. Nguyen
Reg. No. 42,034
Tel.: (714) 557-3800 (Pacific Coast)

12400 Wilshire Boulevard, Seventh Floor
Los Angeles, California 90025

CERTIFICATE OF MAILING/TRANSMISSION (37 CFR 1.8A)

I hereby certify that this correspondence is, on the date shown below, being:

MAILING

☐ *deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, PO Box 1450, Alexandria, VA 22313-1450.*

Date: September 29, 2004

FACSIMILE

☒ *transmitted by facsimile to the Patent and Trademark Office.*

September 29, 2004

Tu Nguyen *Date*